

Prior Art Literature A
(JP-B-H05-052972)

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 03-122729

(43)Date of publication of application : 24.05.1991

(51)Int.Cl.

G06F 12/00

(21)Application number : 02-235953

(71)Applicant : INTERNATL BUSINESS MACH CORP
<IBM>

(22)Date of filing : 07.09.1990

(72)Inventor : MOHAN CHANDRASEKARAN
OBERMARCK RONALD L
TREIBER RICHARD K

(30)Priority

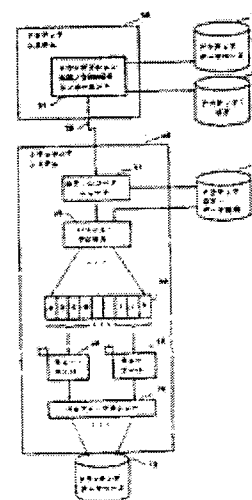
Priority number : 89 411729 Priority date : 25.09.1989 Priority country : US

(54) SYSTEM AND METHOD FOR UPDATING DATA BASE

(57)Abstract:

PURPOSE: To improve parallelism, to efficiently use a change content and to guarantee the consistency of a replica data base and an original data base by using peculiar ordering for recovery records on a transaction log.

CONSTITUTION: The change processing of the replica data base is realized by dividing the REDO (re-issued) records from the transaction log of the primary data base into queues. The REDO records are divided so that all the transaction records in the transfer unit (page) of the primary data base are registered in the same queues 68 and 69 in a log sequence. The queue servers 68 and 69 apply the REDO records in the queues which the servers exclusively handle to the replica data base. Thus, the replica data base becomes consistent with primary data with a lock free update mechanism for processing the pages in parallel.



⑫ 特 許 公 報 (B 2)

平5-52972

⑬ Int. Cl.⁵

G 06 F 12/00

識別記号

5 3 3 J
5 3 1 D

庁内整理番号

7232-5B
7232-5B

⑭ 公告 平成5年(1993)8月6日

請求項の数 5 (全11頁)

⑮ 発明の名称 データベース更新システム及び方法

⑯ 特 願 平2-235953

⑰ 公 開 平3-122729

⑱ 出 願 平2(1990)9月7日

⑲ 平3(1991)5月24日

優先権主張 ⑳ 1989年9月25日㉑ 米国(US)㉒ 411729

⑳ 発 明 者 チヤンドラセカラン・ アメリカ合衆国カリフォルニア州サン・ノゼ、ボーツウッド・ドライブ727番地

㉑ 発 明 者 ロナルド・エル・オバ マーリン・コート584番地

㉒ 発 明 者 リチャード・ケイ・ト レーバー イ・コート5018番地

㉓ 出 願 人 インターナショナル・ アメリカ合衆国10504、ニューヨーク州 アーモンク(番地なし)
ズ・コーポレーション

㉔ 代 理 人 弁理士 頓 宮 孝一 外1名

審 査 官 赤 川 誠 一

㉕ 参 考 文 献 特開 昭63-140352(JP, A) 特開 昭58-166459(JP, A)

1

2

⑳ 特許請求の範囲

1 レコードが転送単位で格納された第1データベースと、該第1データベースのレプリカである第2データベースと、該第1データベースに加えられた変更内容を表わすREDOレコードのシーケンスが格納されたログとを含むトランザクション処理システムにおいて、該第2データベースを、これと第1データベースとの一貫性を保つように更新するシステムであつて、

複数のキューと、

上記第1データベースの1つの転送単位に対応するREDOレコードがすべてログ・シーケンス内の同じキューに配置されるように、上記REDOレコードを上記キューに配置する分配手段と、

上記各キューのREDOレコードを上記第2データベースに適用するためにそれぞれが1つのキューにリンクされた複数のパラレル・キュー・サーバ手段とを含む、システム。

2 請求項1に記載のデータベース更新システム

であつて、上記分配手段がハッシュ・プロセスであるシステム。

3 請求項1に記載のデータベース更新システムであつて、キュー・サーバ手段にリンクされたキューの中の転送単位についてのREDOレコードによつて表される変更内容を、上記転送単位のレコードに適用することによつて、上記キュー・サーバ手段のそれぞれが上記第2データベースの転送単位を更新するシステム。

10 4 複数のキュー・サーバを含むトランザクション処理システムによつて実行され、第1データベースと該第1データベースのレプリカである第2データベースとの一貫性を保証する方法であつて
上記第1データベースに加えられた変更内容に
15 対応するREDOレコードのシーケンスを累積するステップと、

上記第1データベースの1転送単位のREDOレコードがREDOレコード・キュー1個にしか格納されないようにして、複数のREDOレコード・キ

ユーを維持するステップと、

2個以上の上記キューからのREDOレコードを上記第2データベースに並行して適用するステップとを含み、該キューのいずれか1個からのREDOレコードが、上記トランザクション処理システムの上記キュー・サーバのいずれか1個によつてしか該第2データベースに適用されない、データベース更新方法。

5 ローカル・データベースを更新する方法であつて、

ローカル・データベースが一貫性を保つ対象であるアクティブ・データベースからログ・レコードを受けるステップと、

どの選択された物理ストレージ・ユニットに係するログ・レコードも1個のキューにしか格納されないような、上記ログ・レコードのキューを2個以上生成するステップと、

2個以上のサーバ・プロセスを用いることによつて、2個以上の上記キューからのログ・レコードをローカル・データベースに同時適用するステップとを含み、1個のキューからのログ・レコードが、任意の時点で1個のサーバ・プロセスによつてしか該データベースに適用されない、データベース更新方法。

発明の詳細な説明

A 産業上の利用分野

本発明は、トランザクション処理に関し、特に、プライマリ・データベースとレプリカ・データベースの一貫性を保つために、プライマリ・データベースのトランザクション・ログからのREDO(再実行)レコードを、レプリカ・データベースにパラレルに適用する分野に関する。

B 従来の技術

従来のトランザクション処理では、プライマリ・データベースをリモート・ロケーションで二重化できる。リモート・データベースの目的は、プライマリ・データベースの媒体に障害があつた場合に備えて、プライマリ・データベースのバックアップをとることであろう。2つのデータベースの間で一貫性を保つためには、プライマリ・データベースの処理をレプリカ・データベースに反映しなければならない。レプリカに対してレコード処理を行うときのスループットは、プライマリ・データベースを対象にしたトランザクション

処理システムによつて得られるスループットと少なくとも同程度でなければならない。レプリカの処理にかかるリソースは、トランザクション処理全体にかかる場合よりも少なくする必要がある。データベースを、プライマリ・データベースに対してトランザクション・コンシステントとし、プライマリ・データベースと同一にするためには、シリアライゼーションが必要である。プライマリ・データベースの可用性は、レプリカに対してトランザクション処理を加えることで減少することのないように維持する必要がある。

従来の技術によるデータベース二重化の具体例として、同期型分散データベースを挙げることができる。IBMのプロトタイプ System R STARは、データベースのコピーを2箇所に格納する同期型分散データベース・システムである。ただし、現在の同期型分散データベース・システムでは、データベース相互間で一貫性を保つための処理を行うために、またデータベースのシリアライゼーションをロックとして利用するために、相当数のメッセージを交換しなければならない。

データベース媒体リカバリ処理には、よく知られたフォワード・リカバリがある。このプロセスでは、データベース・ユニットを再構成するために、ログまたはジャーナルに記録されたりカバリ・データによつて前のバージョンが更新される。その際、データベース・ユニットの元のバージョンは、ステープル・ストレージにセーブされる。障害によつてデータベースのコピーが破壊されると、データベースの最新バージョンを再構成するために、そのユニットを最後にセーブしたバージョンがリストアされ、セーブされたバージョンに対する変更内容が、破壊されて失われたバージョンに対する変更の順序と同じ順序でログから適用される。

同期型分散データベースのフォワード・リカバリは、元のトランザクション処理を“ミラー”するものである。プライマリ・データベースの変更内容は、変更が生じたときと同じシーケンスでログされるので、ログ・データをパラレルに処理するためには、まず、最初から存在していたロックを取得し、次にログから、REDOレコードをパラレルに引き渡せばよい。この操作では、1回のリ

5

カバリで、所定のトランザクションに対する全レコードが処理され、そのトランザクションに対するロックは、再構成が終わるまで維持され、終了後に解除される。このリカバリ・ソリューションを制約するのは、ある特定のトランザクションに対するロックをすべて1回のプロセスで取得しなかった場合には、ロックの衝突が起こり、元のシーケンスを再現できなくなることがあるという点である。これは、ロックが取得されるまでのパラレルリズムを制限し、ロックが利用できない場合は

処理効率を低下させる。
米国特許出願書類第07/059666号では、ログ・レコードが、複数のキューとして、リスタート処理の間にデータベースに再適用される。この書類は、異なるデータベース・ユニットの更新は、ログに示された順序と異なる順序で適用でき、その場合でも正確さが失われることはないとしている。

Agrawalは、「マルチプロセッサ・データベース・マシンのパラレル・ロギング・アルゴリズム (A Parallel Logging Algorithm for Multi-Processor Database Machines)」J. DATABASE MACHINES : Fourth International Workshop, 1985年3月、の中で、複数のログに対するログ・レコードのパラレル・ライトを提案しているが、リカバリ処理の間、これらのログが1個の“バックエンド”プロセッサを通してシリアル処理される。Lyonの「二重化データベース・システムの障害防止設計 (Design Considerations in Replicated Database Systems for Disaster Protection)」、IEEE Computer Conference, 1988年春、では、リモート・データベース機構が、アクティブ・データベースのジャーナルから更新内容を抽出し、それをバックアップ用データベースに適用する。

C 課題を解決するための手段

本発明の基本を成すのは、トランザクション・ログに関するリカバリ・レコードに固有の順序づけを活用することによって、変更内容をパラレルリズムを高めて効率よく適用でき、よってレプリカ・データベースと元のデータベースとの一貫性が保証されるという知見である。

D 実施例

第1図は従来のデータベース管理システムで、

6

データベース・レコードのトランザクション処理をサポートする。第1図のデータベース管理システムは、データベース・マネジャー (DBMGR) 10を含む。マネジャー10は、ステープル・ストレージ (DASD 12など) に格納されたデータベースのアクセスと処理を管理する。管理対象のデータベースは、ステープル・ストレージに転送単位ごとに格納される。ここで転送単位は、マネジャーがトランザクション処理のためにステープル・ストレージから取り出す基本単位である。従来の技術では“ページ”を転送単位とするのが普通である。1ページは、物理的な境界を持つ DASD 12上の記憶域である。この記憶域の一つを参照符号13で示した。

データベースの転送単位は、ステープル・ストレージ12から取り出されてアクティブ・メモリのバッファ (BFR) 14にログされる。バッファ14に入ったメモリ・ページ内のレコードが処理され、ページは、ステープル・ストレージ12内のストレージ・セクタに返される。ステープル・ストレージ12とバッファ14との間の転送は、バッファ・マネジャー (BMGR) 16によって処理される。マネジャー16はDBMGR 10の管理下で動作する。

この基本要素はアプリケーション・プログラム18が使用できる。プログラム18はDBMGRに対してトランザクションと目的のレコードを指定する。DBMGR 10は、BMGR 16を呼び出して、目的のレコードを含むページをバッファ14へ転送させる。ページがバッファ14に入ると、アプリケーション18に必要なトランザクションに応じて目的のレコードが処理される。更新されたページは、ステープル・ストレージのセクタに上書きされる。

システムに障害が起こると、バッファのページを変更するトランザクションが終了した後、更新済みページがバッファからステープル・ストレージ12へ戻る前に、第1図のデータベース・システムの動作が停止することがある。DBMGR 10のリカバリ機能が起動して修正処理が行われるのはこの場合である。

リカバリ処理では、トランザクション・ログ20をステープル・ストレージに維持しておく必要がある。トランザクションが終了するごとに、ロ

グ20がログ・レコードとして記録される。ログ・レコードの構成要素にはUNDO(取消)とREDOがある。UNDOは、トランザクションによつて変更される前のデータベース・レコードのレプリカである。REDOは変更後のレコードのコピーである。

第2図に示すとおり、媒体リカバリ・プロセス22は、トランザクション・ログ20のREDOレコードを使つて“フォワード”リカバリを行う。たとえば、仮にレコード信号RAないしRFで示したレコードが、ステープル・ストレージ12に格納されたデータ・セットに含まれる定義済みデータ・ブロックj(DDB_j)に含まれるものとする。また、DDB_jのレコードRBないしRDに対するトランザクションによつて、それらのレコードが更新されるものとする。更新のため、BMGR16はページDDB_jを実メモリのバッファ14へ転送し、そこでレコードRBないしRDが更新されるが、いずれの場合も、レコード・データCBがCXに変更される。これらのレコードが処理されるとき、レコードが更新されたシーケンスでログ・エントリが作られる。レコードが更新されるごとに、UNDOとREDOのレコードがトランザクション・ログ20に書き込まれる。たとえばレコードRBを更新するトランザクション・ログ・エントリには、UNDOレコードが含まれ、更新前のレコード・データを含むフィールド(CB)、レコードを識別するフィールド(RB)、およびレコードを含むステープル・ストレージ・ページのロケーションを識別する相対ブロック番号(RBN)がつく。ログ・レコードのREDOには、レコード・データの更新版を含むフィールド(CX)、ページを識別するフィールド(RBN)、およびレコードを識別するフィールド(RB)が含まれる。レコードRBないしRDを更新するトランザクションが終了したとすると、BMGR16は、ページDDB_jをバッファ14にコピーする。そこでリカバリ・プロセスは、トランザクション・ログのREDOレコードを使い、レコードRBないしRDを更新する。BMGR16は次にバッファ・ページをDDB_jのステープル・ストレージ・ロケーションに書き込む。

媒体リカバリ・プロセス22は、“バックワード”リカバリ・プロセスを伴う。このプロセスで

は、終了前のトランザクションによつて更新されたレコードが、UNDOレコードによつて前の状態に戻される。

5 フォワード・リカバリとバックワード・リカバリの両機能はIBMのデータベース製品で利用できる。この機能は、プログラム・プロダクトIBM DB2とIMS/ESAのデータベース・マネジャーでサポートされている。これらの製品ではREDOレコードに加えてUNDOレコードもログ10 される。トランザクションがREDOレコードABORTSに関連する場合、UNDOレコードからのレコード・データは、REDOレコードとしてログされ、データベースに適用されるので、元の更新内容がレコードに“バックアウト”される。15 IBMプログラム・プロダクトIMS/ESAのデータベース・マネジャーFAST PATHでは、UNDOレコードではログされず、変更内容は、REDOレコードがログされて更新トランザクションが終了するまで、データベースに維持される。20 これに関して従来の技術では、トランザクションが無事に終了したことは、トランザクション・ログのCOMMIT(確約)レコードによつて記録されるCOMMITオペレーションによつて示される。レコードの変更にトランザクションが異常終了した場合、ABORT(打切り)レコードがログ25 に入力される。

本発明を示す第3図は、アクティブ・システムとともに動作してトラッキング・データベースを維持するトラッキング・システムである。トラッキング・データベースは、アクティブ・システムによつて維持されるプライマリ・データベースのレプリカである。アクティブ・システム50は(IBM システム 370などのコンピューティング・システムでは上述のIBMデータベース製品を構成できる)、トランザクション処理とデータベース管理を行うシステム・コンポーネント51を持つ。コンポーネント51は、アクティブ・データベース52とともにアクティブ・ログ53の維持、更新も行う。REDOレコードがアクティブ・トランザクション・ログ53に入力されると、システム50はこれらのレコードを通信コンポーネント55を通してトラッキング・システム60に与える。

本発明を表すのはトラッキング・システム60

である。システム 6 0 は、アクティブ・ログ 5 3 に書き込まれた REDO レコードを、アクティブ・ログ 5 3 のアクティブ・ログ・データ機構 6 2 に入力されたシーケンスで受信するプロセス（ログ・レコード・レシート） 6 1 を含む。機構 6 2 は DASD などのステープル・ストレージで構成できる。ハツシュ・プロセス 6 5 は、アクティブ・ログ・データ機構 6 2 に入力された REDO レコードをシーケンスを追って取得し、それらを複数の REDO レコード・キュー 6 6 に登録する。

ハツシュ・プロセス 6 5 は従来からのものであり、この意味で、除算／剰余プロセスから構成できる。このプロセスは、各 REDO レコードの相対ブロック番号に対して機能し、各レコードに対して新しいキー値を生成する。ハツシュ・プロセス 6 5 が IMS/VS タイプのデータベース管理システムで実行させるとすると、このデータベース・システムには、REDO レコードをキュー 6 6 から入力／出力デバイスへ移動させる AMI（アクセス・メソッド・インターフェース）が含まれる。周知のとおり、IMS AMI がシリアライズする IMS REDO レコードのサブセットは、インデックスに新しいキー値が生成されたことを表す。新しいキー値は、ある特定のインデックスに適用されるので、ハツシュ・プロセス 6 5 は、所定のインデックスに対して“新しいキー”の REDO レコードがすべてキュー 6 6 のいずれか 1 個に登録されるように動作する。

キュー登録機能は、REDO レコード・キュー 6 6 は暗示的であり、何らかの手法を用いることによつて、キュー・サーバ（後述）によるキュー登録解除処理の間に、キュー登録されたレコードを正しく順序づけることができる。ハツシュ・プロセス 6 5 が実行されると、1 個のキューの内容が、2 ページ以上に関連する REDO レコードになることがあるが、本発明では、そのキューにハツシュされるページに関連した REDO レコードがすべてキューに付加される。

また本発明では、REDO レコードがそれぞれのキューに登録され、後に、アクティブ・トランザクション・ログ 5 3 に置かれたのと同じシーケンスで適用される。

本発明は、これを保証するために、キュー 6 6 のいずれか一つに対してキュー・サーバを 1 個し

か提供しない。すなわち、一組のレコード・キュー 6 6 のうち、キュー a, b, c, d を扱のはキュー・サーバ 6 8 だけである。キュー i, j, k を扱うのはキュー・サーバ 6 9 だけである。したがつてレコード・キュー 6 6 の個々のキューに対応するキュー・サーバは 1 個だけである。ただし 1 個のキュー・サーバは、2 個以上のキューからのレコードを処理できる。

キュー・サーバ（サーバ 6 8, 6 9 など）は、キュー 6 6 から REDO レコードを取り出し、それらの REDO レコードを、バッファ・マネジャー 7 0 を通してトラッキング・データベース 7 2 に適用する。従来のチャネル型 I/O は、レコード・キュー 6 6 からの REDO レコードを、トラッキング・データベースを含むステープル・ストレージにバッファする。

REDO レコードは、一度キュー 6 6 に入ると、キュー・サーバによつてトラッキング・データベース 7 2 に適用される。たとえばキュー b の REDO レコードは、キュー・サーバ 6 8 によつて 1 個ずつ取り出され、それぞれのページを更新するのに用いられる。アクティブ・システムで生じたこれらの変更のシーケンスは、レコード転送プロセスからトラッキング・システム 6 0 まで、アクティブ・トランザクション・ログ 5 3 に保存され、またアクティブ・ログ・データセット 6 2 にも保存される。レコードはそのレコード・ログ・シーケンスでレコード・キュー 6 6 にハツシュされる。その結果、転送単位に関連したキュー上の REDO レコード・シーケンスによつて、各転送単位（ページ）の更新処理が時間順に正しく反映される。

第 3 図に示すように、キュー・サーバ 6 8, 6 9 はパラレルに動作する。このパラレル動作は、たとえばマルチ CPU のコンピュータで異なる CPU を用いることによつてサポートできる。キュー・サーバ 6 9 はそれぞれ、それが扱うキューから REDO レコードを取り出して、トラッキング・データベース 7 2 に適用する。ここでまた、キュー・サーバが IMS/VS 環境で動作するとすれば、バッファ・マネジャー 7 0 は、サーバに呼び出されて、サーバが保持している REDO レコードの相対ブロック番号によつて識別されるブロックの現在値を含むバッファを取得する。従来のバ

11

ツファ・マネジャー 7 0 は、そのプール内のブロックを発見するか、それをトラッキング・データベース 7 2 からアクティブ・ストレージへ転送してレコードを適用するかのいずれかである。キュー・サーバは、REDOレコードをバツファ内のページに適用し、そのページをバツファ・マネジャー 7 0 に返す。そしてページをトラッキング・データベース 7 2 内のセクタに書き込む。

REDOレコードがトラッキング・データベース 7 2 に適用される間、データベースは、ある時間にはトランザクションによつて変更された部分だけしかページに適用されないという点で、一貫性が保たれない状態になる。このような不一致をシールドするために維持されるロックがないため、通常のデータベース・サービスは、トラッキング・データベース 7 2 をアクセスできないようにする必要がある。障害リカバリ・アプリケーションの場合、これはデメリットとはならない。障害が起こったときだけ一貫性が保たれていればよく、トラッキング・データベース 7 2 によるリカバリが開始されるからである。他の場合、データベースの一貫性は、アクティブ・システムに用いられるリカバリ・メカニズムを考慮した方法を採用することでいつでも確保できる。これに関して、UNDOレコードがログされないデータベース・システムでは、ハツシュ・プロセス 6 2 がアクティブ・ログ・データベース 6 2 からのREDOレコードを、トランザクションCOMMITが検出されるまで保持していなければならない。そこでハツシュ・プロセスは、トランザクションに関連するREDOレコードをREDOレコード・キュー 6 6 に登録する。データベース 7 2 をトランザクション・コンシステントな状態に保つために、アクティブ・ログ・データの処理は、REDOレコード・キューがすべて空になるまで停止される。

ログ・レコードにUNDOとREDOのレコードが含まれるデータベース・マネジャーの場合、REDOレコードは、それが届いたときにキュー登録され、したがってトラッキング・データベースに適用される。トランザクションの一貫性を保つために、アクティブ・ログ・データの処理は、REDOレコード・キューがすべて空になるまで停止され、コミットされなかつた全トランザクションに対しては、標準バックアウト・ロジックを実

12

行するために、UNDOレコードがキュー 6 6 を通して、コミットされなかつた全トランザクションに対して逆のシーケンスで適用される。

以上の点から分かるように、本発明は、トラッキング・データベースを維持するための他のプロシージャにはない大きなメリットを提供するものである。第 1 に、アクティブ・データベースの更新とトラッキング・データベースの更新との同期をとる必要がない。これにより可用性とパフォーマンスに関して次のようなメリットが得られる。アクティブ・データベースは、トラッキング・データベースが使用できない場合でも更新される。REDOレコードのトラッキング・データベースへの転送は、ブロックするかまたはバッチ処理でき、効率向上が図れる。また、シリアライゼーションも必要でなくなるので、コストの削減と高速化が図れる。代表的なデータベースは、入力または出力が必要なときにウェイトをサポートするため、複数のパラレル・キュー・サーバは、パラレル I/O リソースを得て、キュー登録レコードのトラッキング・データベースへの適用を高速化できる。第 3 図に示すように、本発明の実施するのは容易である。当業者には明らかなように、本発明のパフォーマンスから、パス長が短くなるというメリットも得られる。ロックが必要なく、キュー・サーバのパラレリズムからスループットが向上するからである。

最適実施例

第 4 図は上述の本発明の最適実施例である。第 4 図の REDO レコード 1 0 5 は、プロセス QUEUE__REDO__RECORD (キュー REDO 記録) 1 1 0 に引き渡される。プロセス 1 1 0 は REDO レコード 1 0 5 をキューに引き渡し、元のトランザクション・ログ・シーケンスを保存してデータベースの正しさを保証する。プロセス 1 1 0 はハツシュ・プロシージャを伴う。これは、REDO レコード 1 0 5 のレコード・ブロック番号 (RBN) を、ラベル NUMBER__OF__QUEUES (キュー数) のデータ・オブジェクト 1 2 2 の中の数 n で割る。得られたインデックスは、WORK__QUEUE__TABLE (ワーク・キュー・テーブル) 1 2 3 とともに、特定の WORK__QUEUE__ENTRY (ワーク・キュー・エントリ) 1 2 5 を識別するのに用いられる。インデックス

13

つきのWORK_QUEUE_ENTRY 125は、WORK_QUEUE_ENTRY 125を先頭とする特定のレコード・キュー内の最初と最後のREDOレコード130へのポインタ127, 129を持つ。WORK_QUEUE_ENTRY 125はまた、キュー・サーバ・プロセス175のPROCESS_INFO(プロセス情報)制御ブロックを指示するSERVER_PROCESS(サーバ・プロセス) 128を持つ。キュー・サーバ・プロセス175は、RESUME_STATUS(再開ステータス) フラグ172をサーバのPROCESS_INFO(プロセス情報)制御ブロックにセットすることによって起動される。各キュー・サーバは独自のPROCESS_INFO制御ブロックを持ち、各WORK_QUEUE_ENTRYは、そのSERVER_PROCESSフィールドによって1個のキュー・サーバしか識別しない。

表1にQUEUE_REDO_RECORDプロセス110を示す。

表 1

```

QUEUE_REDO_RECORD
200 QUEUE_REDO_RECORD(redo __
    record).
    /*入力はREDOレコード。このルーチンはログ・レコード・レシート・プロセスの下で実行
    */
201 index=
    REMAINDER(redo __ record. block __
        number, number_of_queues).
202 retry20:
203 temp=
    work_queue_table(index). work_queue
        __entry.last_redo.
204 redo_record.next_redo=temp.
205 CS(temp, POINTER(redo_record),
    work_queue_table(index). work_queue
        __entry.last_redo).
    /*REDOレコードを、これに関連するワーク・キューからのREDOレコードのLIFO(ラストイン・ファーストアウト) チェインに自動的に追求*/
206 IF fail=ON THEN
207 GOTO retry20.
    /*アトミック・オペレーション失敗*/

```

14

```

208 IF work_queue_table(index). work
    queue_entry.server_process Process_info.
    resume_status=OFF THEN
    RESUME(work __ queue __ table(index).
    work_queue_entry.server_process).
    /*このワーク・キューのサーバ・プロセスが
    保留される可能性がある場合、そのプロセスを
    再開して追加ワークを処理*/

```

*209 END QUEUE_REDO_RECORD.

表1は、QUEUE_REDO_RECORDプロセス110を従来の疑似コードでインプリメントしたものである。このプロシージャの機能は、REDOレコードをキュー・サーバ・プロセスに引き渡し、元のトランザクション・ログ・シーケンスを保存することにある。本発明の最適実施例では、REDOレコードがWORK_QUEUE(ワーク・キュー)に登録され、関連のキュー・サーバ・プロセスの実行が保留されているか、またはWORK_QUEUEを調べることなく保留される可能性がある場合に、その実行が再開される。

表1で、ステートメント201は、WORK_QUEUE_TABLE 123のインデックスを得て特定のWORK_QUEUE_ENTRY 125を識別する。インデックスは、REDOレコードRBNをデータ・オブジェクトNUMBER_OF_QUEUES 122の値で割り、余りをインデックスとすることによって生成される。ハッシュ法に必要なのは、同じデータベース・ブロック(ページすなわち転送単位)に変更を加えるREDOレコードがすべて同じハッシュ値を持つことである。このハッシュ値により、これらのレコードが同じWORK_QUEUEに関連づけられる。

ステートメント202ないし207は、周知の比較・スワップ命令を用い、順序づけられたWORK_QUEUEに新しいREDOレコードを登録する。キューの順序づけはLIFOが望ましい。ステートメント208は、WORK_QUEUEに関連したキュー・サーバ・プロセスが、キューに追加されたばかりのワークを検出する前に、実行を保留しているかどうかをチェックする。保留できる場合、OSのRESUME(再開)機能が発行されて、RESUME_STATUSフラグ172がONにセットされ、キュー・サーバ・プロセス175の実行が保留されているか、保留オペレーション

15

であれば、再開される。当業者には明らかなように、比較とスワップといったアトミック命令を用いたワークのキュー登録と再開処理を組み合わせるのは、ワークをプロセスからプロセスへ引き渡すための手法として周知のものである。プロセスの保留と再開は、従来のコンピュータのOSに提供されている標準サービスである。

ここで表 2 に第 4 図の QUEUE_SERVER (キュー・サーバ) 175 を示す。

表 2

```

QUEUE_SERVER
300 QUEUE_SERVER(first_entry_index,
    number_of_entries, Process_info).
/*このルーチンはWORK_QUEUE_TABLE
のワーク・キューからのREDOレコードに適用可能。
このルーチンが起動されるプロセスは、これら
特定のワーク・キューを扱う唯一のプロセス*/
301 last_entry_index = first_entry_index
    + number_of_entries - 1.
302 top.
303 Process_info.resume_status = OFF.
304 DO index = first_entry_index TO last
    _entry_index BY 1.
305 IF work_queue_table(index). work_
    queue_entry.last_redo is not null THEN
306 DO.
    /*ワーク検出。キューにおけるこのバスの終
    わりでは保留しない*/
307 Atomically remove entire list from
    work_queue_table(index). work_queue
    _entry.last_redo.
    /*ここで取り除かれたリストはLIFOオーダ
    */
308 Reorder the list of redo records to
    FIFO order chained off of work_queue_
    table(index). work_queue_entry.first
    redo.
    /*ここでREDOレコードは、それを処理する
    順序でチェインされる*/
309 DO WHILE(work_queue_table
    (index). work_queue_entry.first_redo is
    not equal to null).
310 Remove firstredo_record from work_

```

16

```

queue_table(index). work_queue_entry.
first_redo.
311 block_pointer = OBTAIN_BLOCK
    (redo_record.database_name, redo_
    record.block_number).
/*REDOレコードの変更内容が適用される物
理ブロックのコピー（メイン・メモリ内）への
ポインタを取得*/
312 Alter the block using redo_record.redo
    _data.
313 RELEASE_BLOCK(block_pointer).
/*ブロックをディスク・デバイスなどのステ
ープル・ストレージへ書き込めるようにする
*/
314 END./*ワークを持つことが検出された
キューのREDOレコードを適用。キューをルー
プ*/
315 END.
316 END.
20 317 IF process_info.resume_status = OFF
    THEN
    SUSPEND(process_info.resume_status).
    /*キュー・チェックの間に何もキューに登
    録されなかった場合はワークを待つ*/
25 318 GOTO top.
    /*このプロセスを止めるロジックは記載しな
    い*/
319 END QUEUE_SERVER
    REDOレコードを第 3 図のトラッキング・デー
    タベース 72 に適用するためには、キュー・サー
    バ・プロセスが 1 個以上必要になる。適当な大き
    きのシステムでは、アクティブ・システム 50 が
    レコードを作成するレートに近いレートでレコー
    ドを適用するためには、複数のプロセスが必要に
    なる。従来の技術には、システム・アクティビテ
    イに応じてキュー・サーバ・プロセスの個数をダイ
    ナミックに決定・変更する機能がある。さらに、
    特定のワーク・キューの特定のサーバ・プロセ
    スへの割当を変えれば、従来の機能を使って負
    荷を分散できる。表 2 は、説明の便宜上、キュー
    ・サーバ・ロジックを単純化している。これは、
    システムの初期化時に、NUMBER_OF_
    QUEUE_SERVER_PROCESSES(キュー・サ
    ーバ・プロセス数) 120 が、キューをキュー・

```

17

サーバ・プロセスにインストールして割り当てることによつて選択された値mに固定される状態が取られることを示す。

第4図は、第3図の平行・キュー・サーバ・アーキテクチャは示していないが、図と表2から分かるように、二重化によつて第3図のトラッキング・システム60にパラレルリズムを提供する1個のキュー・サーバの機能と構造を示している。

各キュー・サーバ・プロセスは、ある時点では1個以上のワーク・キューを担当するだけである。そのときキューの個数は、キュー・サーバ・プロセスの個数より多いかまたは同一である。

本発明では、単純なインクリメンタル・ループを使えば、サーバ・プロセスをスタートさせ、ワーク・キュー0からn-1(nはデータ・オブジェクト122の大きさ)の範囲の1個以上のワーク・キューに割り当てることができる。各プロセスは、関連のPROCESS_INFO制御ブロックを持ち、このブロックは、125などのWORK_QUEUE_ENTRYのフィールド128から検出できる。各プロセスは、これに制御が移ると、表2のQUEUE_SERVERを起動し、これに、ワーク・キューのインデックス値0ないしn-1(プロセスに割り当てられたもの)、NUMBER_ENTRIES(エントリ数)値1、およびそのPROCESS_INFO172を引き渡す。

先にも述べたとおり、表2は、キュー・サーバを従来の疑似コードでインプリメントしたものである。このプロシージャの機能は、REDOレコードがすべて適用されたときに、トラッキング・データベースの内容がアクティブ・データベースの内容と同じになるように、REDOレコードを、対応するトラッキング・データベース・ブロックに適用することである。

ステートメント303は、RESUME_STATUS172をOFFにして、後にプロセス110によつてREDOレコードが追加された結果、レコードがキューに追加される前にそのレコードに対してプロセス175がすでにロックされている場合には、プロセス175がRESUMEされるようにする。

ステートメント304は、プロセス175が担当するテーブル125のWORK_QUEUE__

18

ENTRYをそれぞれ調べるため、ループを開始する。ステートメント305は、適用されるREDOレコードのWORK_QUEUE_ENTRYをチェックする。キュー登録されたREDOレコードがあれば、アトミック命令のシーケンスにより、マルチプロセス共有リスト・ヘッダLAST_REDO(最終REDO)127から全レコードが除外される。除外されれば、リストが処理されてレコードのシーケンスが逆になり、非共有リスト・ヘッダFIRST_REDO(先頭REDO)129からのレコードがチェインされる。FIRST_REDO129から始まるレコードのリストはここで、特定のデータベース・ブロック(ページ)に対するREDOレコードを含み、変更内容は、アクティブ・システムに適用されたのと同じシーケンスで特定ブロック(ページ)に適用される。

ステートメント309ないし314は、REDOレコードをシーケンスを追つてリストから除外し、それぞれのREDO_DATA(REDOデータ)フィールド38を、レコードのDATABASE_NAME(データベース名)フィールド134で識別されるデータベースに適用する。ステートメント311では、OBTAIN_BLOCK(ブロック獲得)機能が用いられて、所要データベース・ブロックのメイン・メモリ・コピーが検出される。OBTAIN_BLOCKとRELEASE_BLOCK(ブロック解放)の機能は、これらがバツファのプールを管理し、DASDなどの外部媒体でデータベースのステーブル・コピーをリード/ライトするという点で、代表的なデータベース・システム・バツファ管理コンポーネントである。アクティブ・システム50のロジックは、ロックを使つて、アクティブ・データベース52内のデータベース・ブロックへのアクセスをシリアルライズするが、トラッキング・システム60のロジックは、データベース・ブロックへのアクセスのシリアルライゼーションを必要としない。そのデータベース・ブロックをアクセスするキュー・サーバ・プロセスは1個だけだからである。ステートメント312は、変更内容をREDO_DATAフィールド138からデータベース・ブロックのメモリ・コピーに適用する。

ステートメント317では、QUEUE_REDO_RECORDプロセス110がREDOレコードを

19

キューに追加登録しておらず、キュー・サーバ・プロセスをRESUME(表1のステートメント208)している場合、キュー・サーバ・プロセスがステートメント203でRESUME__STATUSをOFFにしているのを、OSの
5 SUSPEND(中断)機能が起動される。RESUMEがすでに発行されているか、または発行されたときは、ステートメント318が再び処理されて、表2のキュー・サーバが、ステートメント302までをループすることによってキュー
10 の処理を継続する。

E 発明の効果

レプリカ・データベースの変更処理は、プライマリ・データベースのトランザクション・ログからのREDOレコードをキューに分けることによつて達成される。REDOレコードの分割は、プライマリ・データベースの転送単位(ページ)のトラ

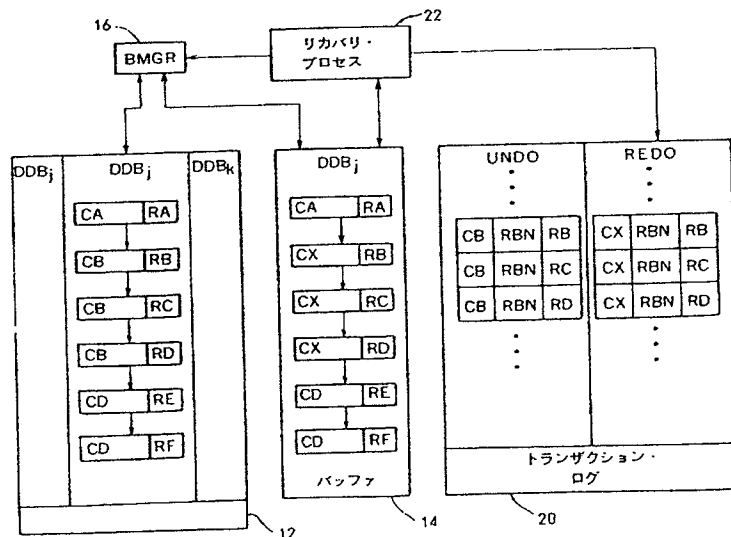
20

ンザクション・レコードがすべて、ログ・シーケンス内の同じキューに登録されるように実行される。各キュー・サーバは、これが排他的に取り扱うキュー内のREDOレコードを、レプリカ・データベースに適用する。これによりレプリカ・データベースは、そのページを平行に処理するロック・フリー更新メカニズムによつて、プライマリ・データと一貫したものとなる。

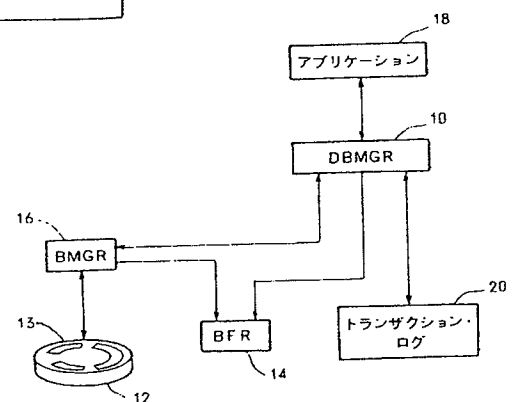
図面の簡単な説明

第1図は、データベース管理システムとしての従来のトランザクション処理システムを示す図である。第2図は、第1図のデータベース管理システムの詳細を示すブロック図である。第3図は、第1図に示す従来のデータベース管理システムと本発明との連結を示すブロック図である。第4図は、本発明で用いられるキューの構造を示す図である。

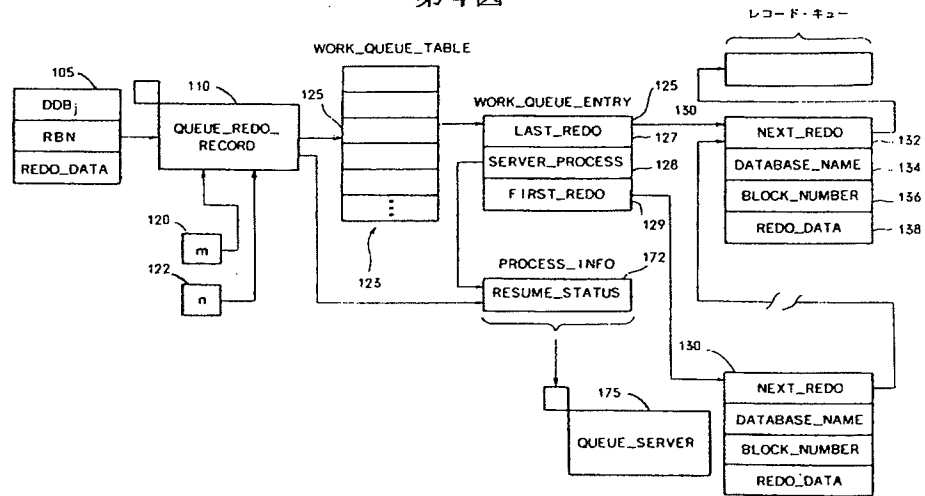
第2図



第1図



第4図



第3図

